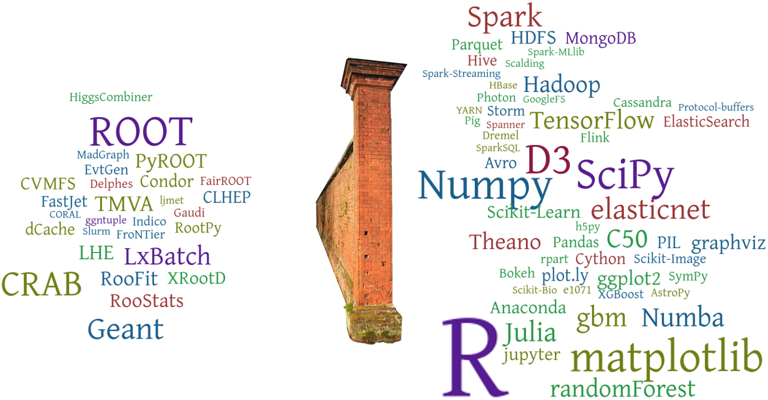
Lowering boundaries between data analysis ecosystems

Jim Pivarski

Princeton University – DIANA Project

May 3, 2017

Data analysis ecosystems



Physicists developed their own software for a good reason: no one else was tackling such large problems.

Not so today. . .


CERN Accelerating science Sign in Directory

About CERN | Students & Educators | Scientists | CERN community | English | Français

Accelerators | Experiments | Physics | Computing | Engineering | Updates | Opinion

CERN Data Centre passes 100 petabytes

by Cian O'Luanaigh

Posted by Cian O'Luanaigh on 14 Feb 2013. Last updated 12 Mar 2015, 13.40.
Voir en français

This content is archived on the CERN Document Server

Servers at the CERN Data Centre collected 75 petabytes of LHC data in the last three years, bringing the total recorded physics data to over 100 petabytes (Image: CERN)

Computer engineers at CERN today announced that the CERN Data Centre has recorded over 100 petabytes of physics data over the last 20 years. Collisions in the Large Hadron Collider (LHC) generated about 75 petabytes of this data in the past three years.

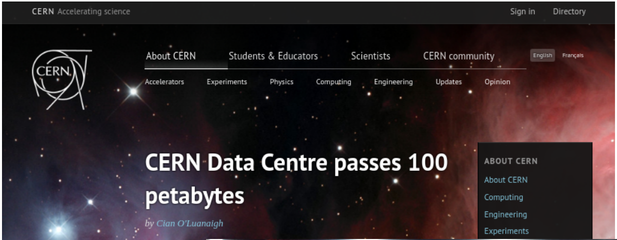
One hundred petabytes (which is equal to 100 million gigabytes) is a very large number indeed – roughly equivalent 700 years of full HD-quality movies. Storing it is a challenge. At CERN, the bulk of the data (about 88

ABOUT CERN
About CERN
Computing
Engineering
Experiments
How a detector works
more ›

CERN UPDATES
The LHC has restarted for its 2017 run
28 Apr 2017
Who switches on the LHC?
28 Apr 2017
CERN and American Physical Society sign SCOAP3
28 Apr 2017
more updates ›

Search this site | Search

Not so today. . .


CERN Accelerating science Sign in Directory

About CERN | Students & Educators | Scientists | CERN community | English | Français

Accelerators | Experiments | Physics | Computing | Engineering | Updates | Opinion

CERN Data Centre passes 100 petabytes

by Cian O'Luanaigh

ABOUT CERN
About CERN
Computing
Engineering
Experiments

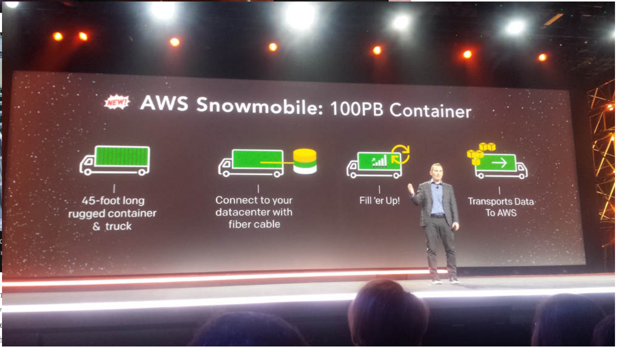
Posted by Cian O'Luanaigh on 14 Feb 2013. Last updated 12 Mar 2015, 13.40.
Voir en français

This content is archived on the CERN Document Server

Servers at the C…
three years, bri…
(Image: CERN)

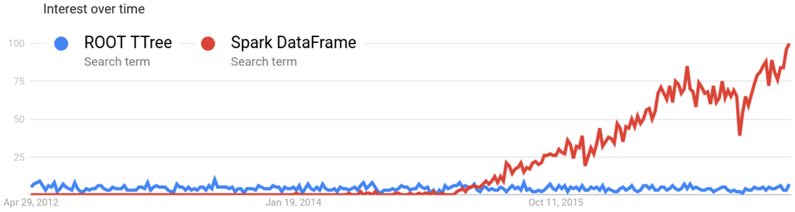
Computer engin…
has recorded ov…
Collisions in the…
of this data in th…

One hundred petabytes (which is equal to 100 million gigabytes) is a very large number indeed – roughly equivalent 700 years of full HD-quality



Case in point: ROOT and Spark

Relative rate of web searches (Google Trends):

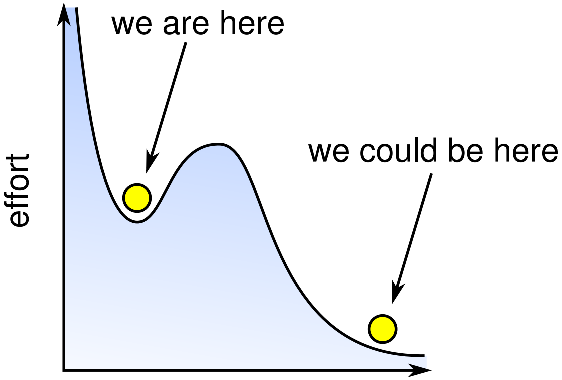


Question-and-answer sites:

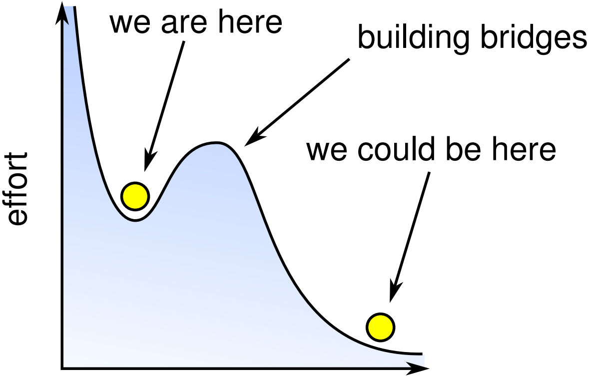
- RootTalk: 14,399 threads in 1997–2012 (15 years)
- StackOverflow questions tagged `#spark`: 26,155 in the 3.3 years the tag has existed.

More users to talk to; more developers adding features/fixing bugs.

Building bridges: low effort-to-reward



Building bridges: low effort-to-reward



Jim Pivarski

- 5 years CLEO (9 GeV e^+e^-)
- 5 years CMS (7 TeV pp)
- 5 years Open Data Group
- 1+ years Project DIANA-HEP

Who am I?

Jim Pivarski

- 5 years CLEO (9 GeV e^+e^-)
- 5 years CMS (7 TeV pp)
- 5 years Open Data Group $\longrightarrow$
- 1+ years Project DIANA-HEP

hyperspectral imagery
automobile traffic
network security
Twitter sentiment
Google n-grams
DNA sequence analysis
credit card fraud detection
and "Big Data" tools

Collaborative Analyses

Establish infrastructure for a higher-level of collaborative analysis, building on the successful patterns used for the Higgs boson discovery and enabling a deeper communication between the theoretical community and the experimental community

Reproducible Analyses

Streamline efforts associated to reproducibility, analysis preservation, and data preservation by making these native concepts in the tools

Interoperability

Improve the interoperability of HEP tools with the larger scientific software ecosystem, incorporating best practices and algorithms from other disciplines into HEP

Faster Processing

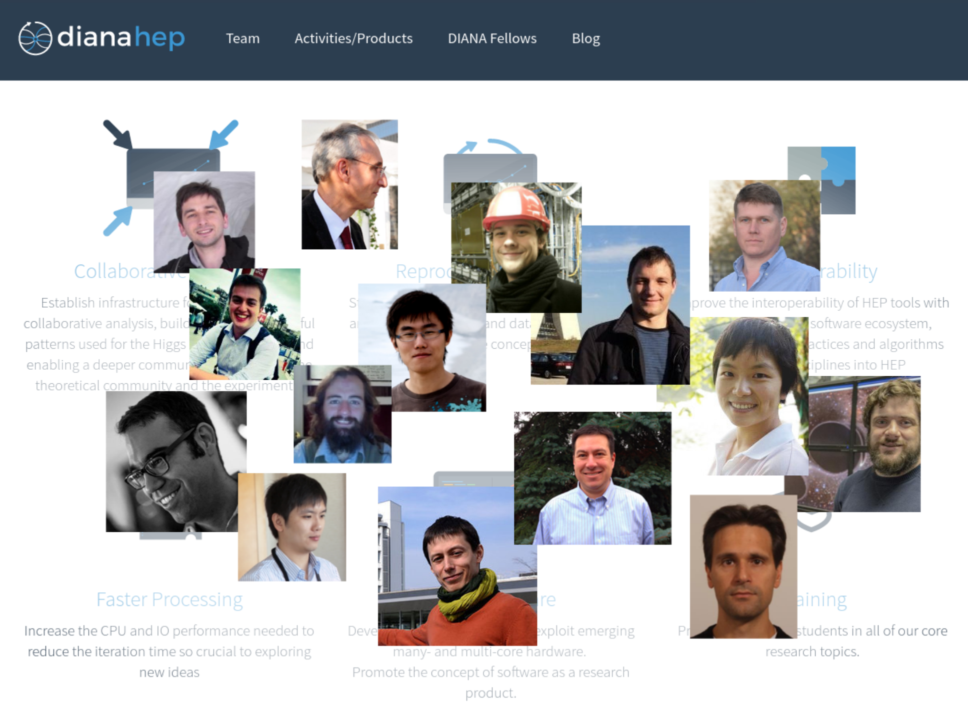
Increase the CPU and IO performance needed to reduce the iteration time so crucial to exploring new ideas

Better Software

Develop software to effectively exploit emerging many- and multi-core hardware.
Promote the concept of software as a research product.

Training

Provide training for students in all of our core research topics.

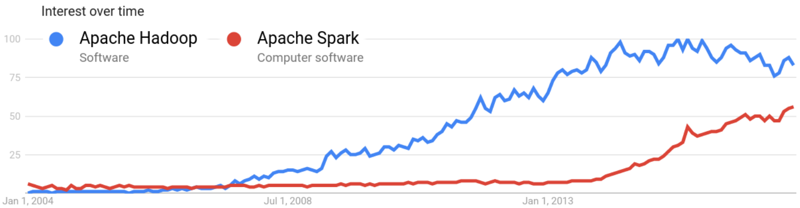

dianahep
Team
Activities/Products
DIANA Fellows
Blog
Collabora
Establish infrastructure f
collaborative analysis, build
patterns used for the Higgs
enabling a deeper commun
theoretical community and the experiment
Repro
and dat
conce
rability
prove the interoperability of HEP tools with
software ecosystem,
actices and algorithms
iplines into HEP
Faster Processing
Increase the CPU and IO performance needed to
reduce the iteration time so crucial to exploring
new ideas
re
Deve
xploit emerging
many- and multi-core hardware.
Promote the concept of software as a research
product.
aining
Pr
students in all of our core
research topics.

Outline of this talk

Data plumbing: a CMS analysis in Apache Spark

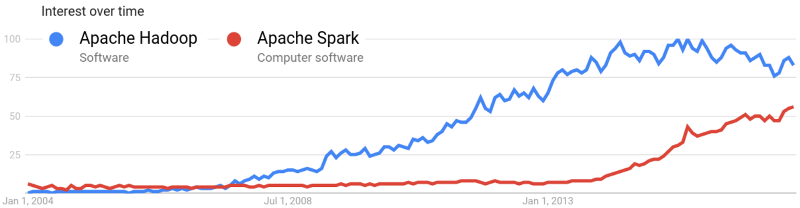
Histogrammar: HEP-like tools in a functional world

Femtocode: the “query system” concept in HEP



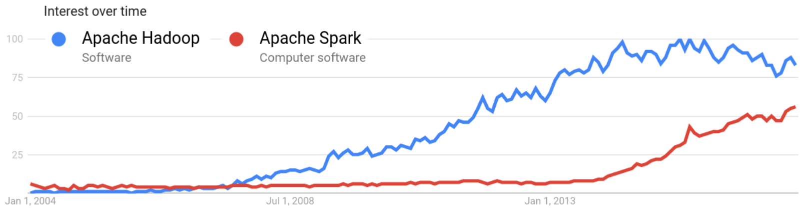
- Like Hadoop in that it implements map-reduce, but these are just two out of many functionals.

Apache Spark



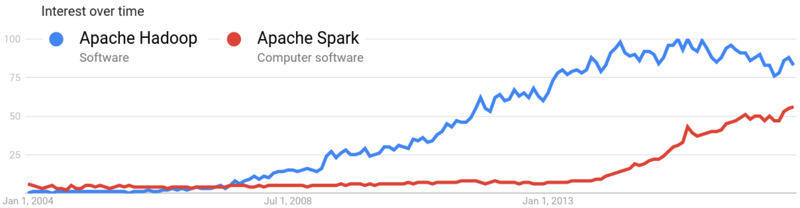
- Like Hadoop in that it implements map-reduce, but these are just two out of many functionals.
- Not a competitor to Hadoop: can run on a Hadoop cluster.

Apache Spark



- Like Hadoop in that it implements map-reduce, but these are just two out of many functionals.
- Not a competitor to Hadoop: can run on a Hadoop cluster.
- Primary interface is a commandline console. Each command does a distributed job and returns a result, While-U-Wait™.

Apache Spark



- Like Hadoop in that it implements map-reduce, but these are just two out of many functionals.
- Not a competitor to Hadoop: can run on a Hadoop cluster.
- Primary interface is a commandline console. Each command does a distributed job and returns a result, While-U-Wait™.
- User controls in-memory cache on the cluster, effectively getting an $\mathcal{O}$(TB) working space in RAM.

CMS analysis on Spark

- Oliver Gutsche, Matteo Cremonesi, Cristina Suárez (Fermilab) wanted to try their CMS dark matter search on Spark.
- This was my first project with DIANA-HEP: I joined to plow through technical issues before the analysts hit them.

https://cms-big-data.github.io/

1. Need a Spark cluster.
2. Spark, like most "Big Data" tools, runs on the Java Virtual Machine (JVM), not C++, and doesn't recognize our ROOT data format.
3. HEP analysis tools like histograms don't have the right API to fit Spark's functional interface.

#1. Need a Spark cluster

Several other groups are interested in this and were willing to share resources in exchange for having us test their system.

- Alexey Svyatkovskiy (Princeton) was active in the group, helping us use the Princeton BigData cluster.
- Saba Sehrish and Jim Kowalkowski (Fermilab) modified the analysis for NERSC.
- Maria Girone, Luca Canali, Kacper Surdy (CERN), and Vaggelis Motesnitsalis (Intel) are now setting up a Data Reduction Facility at CERN as an OpenLab project.
- Offer from Marco Zanetti and Mauro Morandin at Padua.

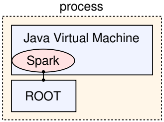
#2. Getting data from ROOT files into JVM

dianahep

A run-down of the attempted solutions...

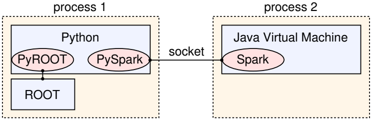
1. **Java Native Interface (JNI)**

   No! This ought to be the right solution, but Java and ROOT are both large, complex applications with their own memory management: couldn't keep them from interfering (segmentation faults).



2. **Python as glue: PyROOT and PySpark in the same process**

   PySpark is a low-performance solution: all data must be passed over a text-based socket and interpreted by Python.



3. **Convert to a Spark-friendly format, like Apache Avro**

   We used this for a year. Efficient after conversion, but conversion step is awkward. Avro's C library is difficult to deploy.

4. **Use pure Java code to read ROOT files**

   What we do now. It's worth it.

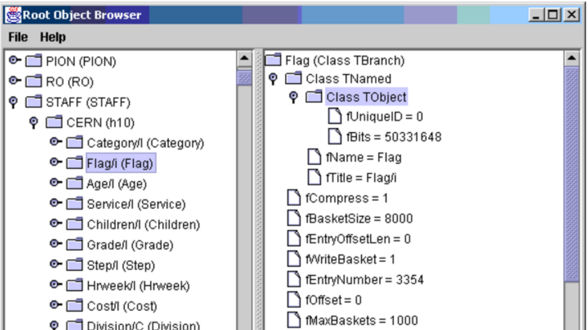
FreeHEP ROOTIO

Root Object Browser

As an illustration of the use of the Java interface, we have built a sample application which is a simple Root Object Browser. It can be used to open any Root file and look at all the objects inside the file. If you already have Java 2 installed (JDK 1.3), you can download the root.jar file containing the application, and run it using the command:

```
java -jar root.jar
```

(on Windows you can just double-click on the root.jar file). A screen shot of the application is show below. The pane on the left shows the directory structure of the file. The object browser knows how to navigate directories (TDirectories), trees (TTrees and TBranches) and these will all be shown in the left pane. Clicking on any object in the left pane will cause the details of the object to be shown in the right pane. The right pane knows how to follow embedded pointers to other objects.



This repository | Search | Pull requests | Issues | Gist

diana-hep / **root4j**

Watch 10 | Star 2 | Fork 2

Code | Issues 1 | Pull requests 0 | Projects 0 | Wiki | Pulse | Graphs | Settings

A fork of http://java.freehep.org/freehep-rootio/ with hooks for Spark DataFrames

Edit

Add topics

45 commits | **2** branches | **2** releases | **2** contributors | LGPL-2.1

Branch: **master** | New pull request | Create new file | Upload files | Find file | Clone or download

vkhristenko making hadoop as provided dependency — Latest commit 2a7bd47 on Mar 15

src	fixing issues with string and other minor updates	3 months ago
.gitignore	updating gitignore	6 months ago
DATAFORMATS.md	updating data format description	4 months ago
LICENSE	Initial commit	6 months ago
README.md	updated readme	6 months ago
pom.xml	making hadoop as provided dependency	2 months ago

README.md

ROOT4J

A fork of http://java.freehep.org/freehep-rootio/

This repository | Search | Pull requests | Issues | Gist

diana-hep / **root4j**

Watch 10 | Star 2 | Fork 2

Code | Issues 1 | Pull requests 0 | Projects 0 | Wiki | Pulse | Graphs | Settings

A fork of http://java.freehep.org/freehep-rootio/ with hooks for Spark DataFrames

Edit

Add topics

45 commits | **2** branches | **2** releases | **2** contributors | LGPL-2.1

Branch: **master** | New pull request | Create new file | Upload files | Find file | Clone or download

vkhristenko making hadoop as provided dependency — Latest commit 2a7bd47 on Mar 15

src	fixing issues with string and other minor updates	3 months ago
.gitignore	updating gitignore	6 months ago
DATAFORMATS.md	updating data format description	4 months ago
LICENSE	Initial commit	6 months ago
README.md	updated readme	6 months ago
pom.xml	making hadoop as provided dependency	2 months ago

README.md

ROOT4J

A fork of http://java.freehep.org/freehep-rootio/

Viktor Khristenko

University of Iowa

Problem #3. Histogram interface

This is how Spark processes data (functional programming):

```
val final_counter =
    dataset.filter(event => event.goodness > 2)
           .map(event => do_something(event.muons))
           .aggregate(empty_counter)(
      (counter, result) => increment(counter, result),
      (c1, c2) => combine(c1, c2))
```

Problem #3. Histogram interface

This is how Spark processes data (functional programming):

```
val final_counter =
    dataset.filter(event => event.goodness > 2)
           .map(event => do_something(event.muons))
           .aggregate(empty_counter)(
      (counter, result) => increment(counter, result),
      (c1, c2) => combine(c1, c2))
```

Read as a pipeline from top to bottom:

1. Start with `dataset` on the cluster somewhere.
2. Filter it with `event.goodness > 2`.
3. Compute `do_something` on each event's muons.
4. Accumulate some counter (e.g. histogram or other data summary), starting with `empty_counter`, using `increment` to fill with each event's `result`, combining partial results with `combine`.

all distributed across the cluster, returning only `final_counter`.

Problem #3. Histogram interface

This is how ROOT/PAW/HBOOK histograms expect to be called:

```
// on a worker handling one partition of data
hist = new TH1F("name", "title", numBins, low, high);

for (i = start_partition;  i < end_partition;  i++) {
    dataset.GetEntry(i);
    if (goodness > 2)
        hist->Fill(do_something(muons));
}

// on the head node, after downloading partial hists
hadd(hists);
```

Problem #3. Histogram interface

Trying to wedge the square peg into the round hole:

```
import ROOT
empty_hist = ROOT.TH1F("n", "t", numBins, low, high)

def increment(hist, result):
    hist.Fill(result)
    return hist

def combine(h1, h2):
    return h1.Add(h2)

filled_hist =
  data.filter(lambda event: event.goodness > 2)      \
      .map(lambda event: do_something(event.muons)) \
      .aggregate(empty_hist, increment, combine)
```

It's not impossible, but it's awkward.

Awkward is bad for data analysis because you really should be focusing on the complexities of your analysis, not your tools.

Making histograms functional

There's a natural way to do histograms in functional programming: add a fill rule to the declaration.

```
hist = Histogram(numBins, low, high,
                     lambda event: event.what_to_fill)
```

Making histograms functional

There's a natural way to do histograms in functional programming: add a fill rule to the declaration.

```
hist = Histogram(numBins, low, high,
                 lambda event: event.what_to_fill)
```

This way, `what_to_fill` doesn't have to be specified in the (non-existent) "for" loop.

```
dataset.fill_it_for_me(hist)
```

It's cooler this way

Functional programming emphasizes composition: building new functionality by composing functions.

```
# standard 1-D histogram
Bin(numBins, low, high, x_rule, Count())
```

- `Bin` splits into bins by `x_rule`, passes to a `Count` in each bin,
- `Count` counts.

It's cooler this way

Functional programming emphasizes composition: building new functionality by composing functions.

```
# profile plot
Bin(numBins, low, high, x_rule, Deviate(y_rule))
```

- `Bin` splits into bins by `x_rule`, passes to a `Deviate` in each bin,
- `Deviate` computes the mean and standard deviation of `y_rule`.

Functional programming emphasizes composition: building new functionality by composing functions.

```
# 2-D histogram
Bin(numBins, low, high, x_rule,
    Bin(numBins, low, high, y_rule,
        Count()))
```

- `Bin` splits into bins by `x_rule`, passes to a `Bin` in each bin,
- second `Bin` does the same with `y_rule`.

It's cooler this way

Functional programming emphasizes composition: building new functionality by composing functions.

```
# different binning methods on different dimensions
Categorize(event_type,
    SparselyBin(trigger_bits,
        IrregularlyBin([-2.4, -1.5, 1.5, 2.4], eta,
            Bin(100, 0, 100, pt,
                Count())))))
```

- `Categorize` splits based on string value (like a bar chart)
- `SparselyBin` only creates bins if their content is non-zero
- `IrregularlyBin` lets you place bin edges anywhere

It's cooler this way

Functional programming emphasizes composition: building new functionality by composing functions.

```
# bundle histograms to be filled together
Bundle(
    one = Bin(numBins, low, high, fill_one),
    two = Bin(numBins, low, high, fill_two),
    three = Bin(numBins, low, high, fill_three))
```

- `Bundle` is a directory mapping names to aggregators; same interface as all the other aggregators

It's cooler this way

Functional programming emphasizes composition: building new functionality by composing functions.

```
# to organize your analysis

pack_o_plots = Bundle(
    one = Bin(numBins, low, high, fill_one),
    two = Bin(numBins, low, high, fill_two))

Bundle(
    withcut = Select(cut_rule, pack_o_plots),
    nocut   =                  pack_o_plots)
```

- `Select` only passes down events that pass `cut_rule`
- `Bundles` are now nested like subdirectories, one `pack_o_plots` with cut, the other without

It's cooler this way

Functional programming emphasizes composition: building new functionality by composing functions.

```
# or do wacky things
Bin(numBins, low, high, lambda event: event.x,
    Bundle(
        nonzero = Fraction(lambda event: event.y > 0,
                           Count()),
        mean = Average(lambda event: event.y),
        maximum = Maximize(lambda event: event.y)))
```

- fills a *directory* of "nonzero," "mean," and "maximum" *in each bin.*

dianahep DOCUMENTATION GITHUB

histo·*grammar*

/histōˌˈɡɹæm.ər/

MAKING HISTOGRAMS FUNCTIONAL

Histogrammar is a suite of data aggregation primitives for making histograms and much, much more. A few composable functions can generate many different types of plots, and these functions are reimplemented (exactly!) in multiple languages and serialized to JSON for cross-platform compatibility.

http://histogrammar.org

Femtocode: query system for HEP

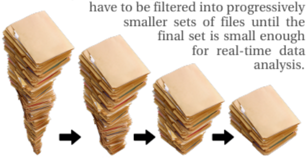
What's a query system?

User asks a question, gets an answer quickly enough to *explore* the data.

Like a Google query, but aggregating HEP data, returning (e.g.) histograms.

Embedded within an analysis script: provides sliced projections of the data for users to fit/plot/analyze in any way they want.

How it differs from what we do now



Physicists arrange data as sets of files that have to be filtered into progressively smaller sets of files until the final set is small enough for real-time data analysis.

Instead, we propose a service that serves aggregated views of analysis object data on demand.

Must be responsive to requests in real-time: ~1 sec for each scan over a dataset.

High-level language

User writes expressions that pick apart the structure of objects within arbitrary-length lists, to any depth of nesting.

Higher-order functions like `.map`, `.pairs`, `.filter`, `.reduce` instead of explicit `for` loops.

Femtocode query language is distributed in quoted snippets throughout a structured workflow and tree of aggregators.

(See `http://histogrammar.org` for histogram abstraction.)

```
workflow = session.source("b-physics")                       # pull from a named dataset
        .define(goodmuons = "muons.filter($1.pt > 5)")       # muons with pt > 5 are good
        .filter("goodmuons.size >= 2")                       # keep events with at least two
        .define(dimuon = """
            mu1, mu2 = goodmuons.maxby($1.pt, 2);            # pick the top two by pt
            energy = mu1.E + mu2.E;                          # compute imploded energy/momentum
            px = mu1.px + mu2.px;
            py = mu1.py + mu2.py;
            pz = mu1.pz + mu2.pz;
```

Low-level execution

No objects at runtime:

All nested structures are represented as homogeneous arrays.

```
type           = collection(collection(record(a=integer, b=real)))
values         = [ [(1, 1.1)] [] [(2, 2.2) (3, 3.3)]] [ [(4, 4.4)]]
```

becomes

```
data[][]@size  = 3 1         0  2               1 1
data[][]-a     =     1           2       3          4
data[][]-b     =         1.1          2.2    3.3        4.4
```

```
muons.map({mu1 =>
  muons.map({mu2 =>
    e1 = mu1.p**2 + 0.105658**2;  } physicist
    e2 = mu2.p**2 + 0.105658**2;  } unnecessarily
    e1 + e2                         wrote these
    }).max                          lines in the
                                    loop over
                                    muon pairs
```

Wrap-up

- We're not the big fish anymore: time to look to industry to see how they're solving problems similar to ours.
- Historical mismatches in non-essential details (e.g. data formats) are annoying, but surmountable.
- Differences in fundamental approach are an opportunity: alien civilizations can learn from each other.